Augmented Democracy: Technical Appendices

Formal Definitions, Protocol Mechanics, Security Analysis, and Implementation

Sylvain Cormier
sylvain@paraxiom.io

December 2025

Abstract

This document provides the technical appendices for "Augmented Democracy: Procedural Coherence in Algorithmically Mediated Governance." It contains formal definitions, mathematical proofs, protocol specifications, security analysis, and implementation details that support the main paper. These appendices are intended for implementers, auditors, and researchers requiring full technical depth.

Contents

A Formal Definitions

This appendix provides rigorous mathematical definitions for all constructs introduced in the main paper.

A.1 Core Spaces and Structures

Definition A.1 (Participant Space)**.** *Let $\mathcal{P}$ denote the finite set of registered participants in the governance system. Each participant $p \in \mathcal{P}$ is identified by a unique cryptographic identity $id_p \in \{0,1\}^{256}$ derived from their public key.*

Definition A.2 (Proposal Space)**.** *The proposal space Π consists of all well-formed governance proposals. A proposal $\pi \in \Pi$ is a tuple:*

$$\pi = (id_\pi, \mathit{content}, \mathit{proposer}, \mathit{timestamp}, \mathit{artifacts}, \mathit{status})$$

where:

- $id_\pi \in \{0,1\}^{256}$ *is the proposal hash*
- *content is the proposal description and code (if applicable)*
- $\mathit{proposer} \in \mathcal{P}$ *is the submitting participant*
- $\mathit{timestamp} \in \mathbb{N}$ *is the block number of submission*
- $\mathit{artifacts} \subseteq \mathcal{A}$ *is the set of referenced fact artifacts*
- $\mathit{status} \in \{\mathit{pending}, \mathit{active}, \mathit{passed}, \mathit{rejected}\}$

Definition A.3 (Admissible Fact Artifact)**.** *An admissible fact artifact is a tuple $a = (h, I, R)$ where:*

- $h \in \{0,1\}^{256}$ *is a cryptographic hash or DOI providing content-addressability*
- $I : \mathcal{P} \to \{0,1\}$ *is an issuance predicate indicating authentic origin*
- $R : \Pi \to \{0,1\}$ *is a relevance predicate indicating contextual applicability*

An artifact a is admissible for proposal π if and only if $I(a.\mathit{issuer}) = 1$ and $R(\pi) = 1$.

Definition A.4 (Credential Function)**.** *The credential function $r : \mathcal{P} \times \mathbb{N} \to [0,1]$ maps each participant-epoch pair to a reputation score. The function must satisfy:*

1. ***Boundedness**: $\forall p, t : 0 \leq r(p,t) \leq 1$*
2. ***Decay**: $r(p, t+1) \leq r(p,t) + \delta_{\max}$ where $\delta_{\max}$ is the maximum per-epoch gain*
3. ***Initial neutrality**: $r(p,0) = r_0$ for some constant $r_0 \in (0,1)$*

Definition A.5 (Vote Weight Function)**.** *The vote weight function $w : \mathcal{P} \times \Pi \to \mathbb{R}_{\geq 0}$ is defined as:*

$$w(p, \pi) = r(p, t_\pi) \cdot (1 + \epsilon_p)$$

where t_π is the epoch containing π and $\epsilon_p \sim \mathit{Uniform}(-0.1, 0.1)$ is a per-participant entropy injection sampled from quantum random sources.

A.2 Test Grid Definitions

Definition A.6 (Token-Curated Test Grid)**.** *A Token-Curated Test Grid is a tuple $G = (Q, C, S, \tau)$ where:*

- $Q = \{q_1, \ldots, q_n\}$ *is a set of n engagement verification questions*
- $C \subseteq \mathcal{P}$ *is the set of curators who maintain Q*
- $S : C \to \mathbb{R}_{\geq 0}$ *is the stake function mapping curators to bonded tokens*
- $\tau \in (0, 1]$ *is the passing threshold for engagement verification*

Definition A.7 (Engagement Score)**.** *For participant p and test grid G, the engagement score $e(p, G)$ is:*

$$e(p, G) = \frac{|\{q \in Q : \textit{correct}(p, q)\}|}{|Q|}$$

Participant p passes engagement verification if and only if $e(p, G) \geq \tau$.

Definition A.8 (Curator Slashing Condition)**.** *A curator $c \in C$ is slashed (loses stake $S(c)$) if and only if one of the following procedural violations is proven:*

1. ***Inaccessibility****: Question q was not retrievable during voting period*
2. ***Ambiguity****: Question q admits multiple defensible correct answers*
3. ***Irrelevance****: Question q is unrelated to the proposal domain $D(\pi)$*

Slashing does not *occur for contested semantic conclusions within the domain.*

A.3 Coherence Definitions

Definition A.9 (Entropy-Weighted Variance)**.** *Let $V = \{v_1, \ldots, v_m\}$ be the set of votes for proposal π, where each vote v_i has weight w_i and entropy contribution η_i. The entropy-weighted variance is:*

$$\sigma_\eta^2 = \frac{\sum_{i=1}^{m} w_i \cdot (v_i - \bar{v})^2 \cdot \eta_i}{\sum_{i=1}^{m} w_i \cdot \eta_i}$$

where $\bar{v} = \frac{\sum_i w_i v_i}{\sum_i w_i}$ is the weighted mean vote.

Definition A.10 (Coherence Score)**.** *The coherence score $\gamma \in [0, 100]$ for a proposal vote is:*

$$\gamma = \frac{\sigma_\eta^2}{255} \times 100$$

where σ_η^2 is the entropy-weighted variance and 255 is the normalization constant (maximum possible variance with 8-bit entropy values).

Definition A.11 (Dual Condition)**.** *A proposal π passes if and only if both conditions hold:*

1. ***Majority condition****: $W_{approve}(\pi) > W_{reject}(\pi)$*
2. ***Coherence condition****: $\gamma(\pi) > \gamma_{\min}$ (default: $\gamma_{\min} = 50$)*

A.4 Governance System Definition

Definition A.12 (Governance Control System)**.** *An Augmented Democracy governance system is a tuple* $\mathcal{G} = (S, A, T, I, R)$ *where:*

- S *is the state space (configuration parameters, treasury, code)*
- A *is the action space (proposals, votes, stake operations)*
- $T : S \times A \to S$ *is the transition function*
- $I : S \times A \to \{0, 1\}$ *is the invariant check (dual condition)*
- $R : S \to \mathbb{R}$ *is the reference trajectory (policy objectives)*

The transition $T(s, a) = s'$ *occurs if and only if* $I(s, a) = 1$*.*

B Protocol Mechanics

This appendix specifies the detailed protocol mechanics for implementation.

B.1 Credential Decay and Accumulation

Credential Update Protocol

At each epoch boundary $t \to t + 1$, for each participant p:

1. **Compute decay**: $r_{\text{decay}}(p) = r(p, t) \cdot (1 - \lambda)$ where $\lambda = 0.05$ is the base decay rate
2. **Compute participation bonus**:

$$\delta_p = \begin{cases} \delta_{\text{vote}} & \text{if } p \text{ voted this epoch} \\ \delta_{\text{propose}} & \text{if } p \text{ proposed this epoch} \\ \delta_{\text{curate}} & \text{if } p \text{ curated correctly} \\ 0 & \text{otherwise} \end{cases}$$

   where $\delta_{\text{vote}} = 0.01$, $\delta_{\text{propose}} = 0.02$, $\delta_{\text{curate}} = 0.03$

3. **Apply slashing** (if applicable):

$$r_{\text{slash}}(p) = \begin{cases} r_{\text{decay}}(p) \cdot 0.5 & \text{if minor violation} \\ 0 & \text{if major violation} \end{cases}$$

4. **Update credential**:

$$r(p, t + 1) = \min(1, r_{\text{decay}}(p) + \delta_p - \text{slash}_p)$$

B.2 Voting Protocol

Vote Submission Protocol

For participant p voting on proposal π:

1. **Retrieve test grid**: Fetch current grid G for domain $D(\pi)$
2. **Engagement verification**:
   - Sample k questions uniformly from Q
   - Present questions to p
   - Compute $e(p, G)$
   - If $e(p, G) < \tau$: reject vote submission
3. **Weight calculation**:
   - Fetch $r(p, t)$ from credential registry
   - Sample ϵ_p from quantum entropy pool
   - Compute $w_p = r(p, t) \cdot (1 + \epsilon_p)$
4. **Vote recording**:
   - Record vote $(p, \pi, v, w_p, \epsilon_p, \text{timestamp})$
   - Emit vote event for transparency

B.3 Proposal Lifecycle

Proposal State Machine

Each proposal π transitions through states:

```
Draft -> Pending -> Active -> (Passed | Rejected | Expired)
```

Transitions:

- `Draft -> Pending`: Proposer submits with required deposit
- `Pending -> Active`: Minimum endorsements reached
- `Active -> Passed`: Dual condition satisfied at voting end
- `Active -> Rejected`: Majority against OR coherence below threshold
- `Active -> Expired`: Voting period ends without quorum

Timing Parameters:

- Review period: 7 days (for amendment and discussion)
- Voting period: 14 days (active voting window)
- Execution delay: 2 days (time-lock after passage)

B.4 Epoch Parameters

Parameter	Symbol	Default Value
Epoch duration	T_{epoch}	28 days
Decay rate	λ	0.05 (5%)
Vote participation bonus	δ_{vote}	0.01
Proposal bonus	δ_{propose}	0.02
Curation bonus	δ_{curate}	0.03
Coherence threshold	$\gamma_{\min}$	50
Engagement threshold	τ	0.7 (70%)
Initial credential	r_0	0.5
Entropy range	ϵ	$\pm 10\%$

Table 1: Default protocol parameters

C Security Analysis

This appendix provides formal security analysis and proofs.

C.1 Threat Model

We consider an adversary $\mathcal{A}$ with the following capabilities:

- Control of up to $f < n/3$ participants (Byzantine assumption)
- Ability to create synthetic identities (Sybil capability)
- Financial resources for bribery (bounded by $B_{\max}$)
- Computational power for pattern analysis
- Network observation capability (but not partition control)

The adversary's goal is to cause the system to:

1. Pass a proposal that should not pass (false positive)
2. Reject a proposal that should pass (false negative)
3. Undermine confidence in the governance process

C.2 Sybil Resistance

Theorem C.1 (Sybil Detection Bound)**.** *Let $\mathcal{A}$ control k Sybil identities voting identically. If $k > \sqrt{n}$ where n is the total voter count, then $\gamma < \gamma_{\min}$ with probability at least $1 - e^{-k^2/2n}$.*

Proof. Identical votes from Sybil identities contribute identical entropy values (or highly correlated values if attempting evasion). The variance computation:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(v_i - \bar{v})^2$$

When k votes are identical with value v_s:

$$\sigma^2 \leq \frac{(n-k)\cdot 1 + k \cdot 0}{n} = \frac{n-k}{n}$$

For $k > \sqrt{n}$, this yields $\sigma^2 < 1 - 1/\sqrt{n}$.

Combined with entropy weighting, the contribution of Sybil votes is:

$$\gamma_{\text{Sybil}} = \frac{k \cdot \sigma^2_{\text{low}} + (n-k)\cdot \sigma^2_{\text{normal}}}{n}$$

By Hoeffding's inequality, the probability that Sybil voting escapes detection is bounded by $e^{-k^2/2n}$. □

C.3 Bribery Resistance

Theorem C.2 (Quadratic Bribery Cost)**.** *To achieve k units of effective voting power through bribery, an adversary must spend at least $\Omega(k^2)$ resources.*

Proof. Under quadratic voting, acquiring n votes costs n^2. If the adversary wishes to achieve total weight W, they can either:

1. Bribe one participant for $\sqrt{W}$ votes at cost W
2. Bribe m participants for $\sqrt{W/m}$ votes each at cost $m \cdot (W/m) = W$

In either case, the cost scales linearly with desired weight, which under standard voting would only require $\sqrt{W}$ bribes. The quadratic mechanism provides a $\sqrt{W}$ amplification in attack cost.

Furthermore, correlated bribery (coordinated voting) triggers coherence detection:

$$P[\text{detection}|\text{bribery}] \geq 1 - \left(\frac{1}{2}\right)^m$$

where m is the number of bribed voters, since their entropy values will show statistical anomalies. □

C.4 Coordination Attack Resistance

Lemma C.3 (Coordination Detection)**.** *Let $C \subseteq \mathcal{P}$ be a coordinating coalition. If $|C| > 0.1n$ and coalition members vote identically, then $\gamma < \gamma_{\min}$ with probability > 0.95.*

Proof. Coordinated voting produces a bimodal or multimodal distribution in entropy space. The variance contribution from the coordinating bloc is:

$$\sigma_C^2 = \frac{|C|}{n} \cdot \text{Var}(\epsilon_C)$$

Since ϵ_i are independent samples from Uniform$(-0.1, 0.1)$, identical voting suppresses intra-coalition variance to near zero. The system detects this as:

$$\gamma_{\text{observed}} < \gamma_{\text{expected}} - 2\sigma_\gamma$$

By Chebyshev's inequality, for $|C| > 0.1n$:

$$P[\gamma < \gamma_{\min}] > 1 - \frac{\text{Var}(\gamma)}{(\gamma_{\text{expected}} - \gamma_{\min})^2} > 0.95$$

□

C.5 Economic Security Analysis

Proposition C.4 (Attack Cost Lower Bound)**.** *The minimum cost to pass a fraudulent proposal with probability > 0.5 is:*

$$C_{attack} \geq \min\left(\frac{n \cdot \bar{s}}{4}, \frac{W_{total}^2}{4 \cdot W_{honest}}\right)$$

where $\bar{s}$ is the average stake per participant and W_{honest} is honest voter weight.

Proof. An attacker must either:

1. Acquire $> n/2$ credentials through Sybil attacks, requiring defeating the identity verification system and maintaining $n/2$ active credentials without triggering coherence detection
2. Bribe $> W_{\text{honest}}$ weight worth of existing participants, costing at least W_{honest}^2 under quadratic scaling plus the premium for evading detection

The coherence gate provides an additional barrier: even with majority weight, $\gamma < \gamma_{\min}$ causes rejection. To evade this, attackers must distribute votes to appear organic, which requires either:

- More participants (increasing C_{attack} linearly)
- Slower coordination (allowing defender response)

□

C.6 What Coherence Does Not Detect

The coherence mechanism has formal limitations:

Remark C.1 (Undetectable Attacks)**.** *The following attack vectors are not detected by γ:*

1. ***Genuine shared preferences**: Large groups with authentically similar views*
2. ***Information cascades**: Organic convergence after public deliberation*
3. ***Gradual ideology shift**: Slow coordination over multiple epochs*
4. ***Semantic manipulation**: Proposals that are procedurally correct but substantively harmful*

These require complementary mechanisms (diverse curation, deliberation periods, human oversight).

D Implementation

This appendix provides reference implementations for core protocol components.

D.1 Substrate Pallet: Credential Management

Listing 1: Credential storage and decay logic

```rust
#[pallet::storage]
pub type Credentials<T: Config> = StorageMap<
    _,
    Blake2_128Concat,
    T::AccountId,
    CredentialInfo<T::BlockNumber>,
    OptionQuery,
>;

#[derive(Encode, Decode, Clone, TypeInfo)]
pub struct CredentialInfo<BlockNumber> {
    pub reputation: u32,        // Fixed-point [0, 10000] = [0.0, 1.0]
    pub last_active: BlockNumber,
    pub participation_count: u32,
    pub slashing_points: u32,
}

impl<T: Config> Pallet<T> {
    pub fn apply_epoch_decay(who: &T::AccountId) -> DispatchResult {
        Credentials::<T>::try_mutate(who, |maybe_cred| {
            let cred = maybe_cred.as_mut()
                .ok_or(Error::<T>::NotRegistered)?;

            // Apply 5% decay
            let decay = cred.reputation / 20;
            cred.reputation = cred.reputation.saturating_sub(decay);

            // Check for inactivity penalty
            let current_block = frame_system::Pallet::<T>::block_number();
            let inactive_epochs = (current_block - cred.last_active)
```

```
                / T::EpochLength::get();

            if inactive_epochs > T::MaxInactiveEpochs::get() {
                cred.reputation = cred.reputation / 2; // Additional
                    penalty
            }

            Ok(())
        })
    }

    pub fn add_participation_bonus(
        who: &T::AccountId,
        action: ParticipationType,
    ) -> DispatchResult {
        let bonus = match action {
            ParticipationType::Vote => 100,      // 0.01
            ParticipationType::Propose => 200,   // 0.02
            ParticipationType::Curate => 300,    // 0.03
        };

        Credentials::<T>::try_mutate(who, |maybe_cred| {
            let cred = maybe_cred.as_mut()
                .ok_or(Error::<T>::NotRegistered)?;

            cred.reputation = cred.reputation
                .saturating_add(bonus)
                .min(10000); // Cap at 1.0
            cred.last_active = frame_system::Pallet::<T>::block_number();
            cred.participation_count += 1;

            Ok(())
        })
    }
}
```

D.2 Substrate Pallet: Coherence Calculation

Listing 2: Coherence score computation

```
pub fn calculate_coherence<T: Config>(
    proposal_id: T::Hash,
) -> Result<u8, Error<T>> {
    let votes = Votes::<T>::iter_prefix(proposal_id)
        .collect::<Vec<_>>();

    if votes.is_empty() {
        return Err(Error::<T>::NoVotes);
    }

    let n = votes.len() as u128;

    // Calculate weighted mean
```

```
    let (total_weight, weighted_sum) = votes.iter()
        .fold((0u128, 0i128), |(tw, ws), (_, vote)| {
            let w = vote.weight as u128;
            let v = if vote.approve { 1i128 } else { -1i128 };
            (tw + w, ws + (w as i128) * v)
        });

    let mean = weighted_sum * 1000 / total_weight as i128;

    // Calculate entropy-weighted variance
    let variance_sum: u128 = votes.iter()
        .map(|(_, vote)| {
            let v = if vote.approve { 1000i128 } else { -1000i128 };
            let diff = (v - mean).abs() as u128;
            let w = vote.weight as u128;
            let e = vote.entropy as u128;

            (diff * diff * w * e) / 1_000_000
        })
        .sum();

    let total_entropy_weight: u128 = votes.iter()
        .map(|(_, v)| (v.weight as u128) * (v.entropy as u128))
        .sum();

    let variance = if total_entropy_weight > 0 {
        variance_sum / total_entropy_weight
    } else {
        0
    };

    // Normalize to [0, 100]
    let gamma = (variance * 100 / 255).min(100) as u8;

    Ok(gamma)
}

pub fn check_dual_condition<T: Config>(
    proposal_id: T::Hash,
) -> Result<bool, Error<T>> {
    let gamma = Self::calculate_coherence::<T>(proposal_id)?;

    if gamma < T::MinCoherence::get() {
        return Ok(false);
    }

    let (approve_weight, reject_weight) =
        Self::tally_weights::<T>(proposal_id)?;

    Ok(approve_weight > reject_weight)
}
```

D.3 Substrate Pallet: Test Grid Verification

Listing 3: Engagement verification logic

```rust
#[pallet::storage]
pub type TestGrids<T: Config> = StorageMap<
    _,
    Blake2_128Concat,
    DomainId,
    TestGrid<T::AccountId, T::Balance>,
    OptionQuery,
>;

#[derive(Encode, Decode, Clone, TypeInfo)]
pub struct TestGrid<AccountId, Balance> {
    pub questions: BoundedVec<QuestionHash, MaxQuestions>,
    pub curators: BoundedVec<(AccountId, Balance), MaxCurators>,
    pub passing_threshold: u8,  // Percentage [0, 100]
    pub last_updated: BlockNumber,
}

impl<T: Config> Pallet<T> {
    pub fn verify_engagement(
        who: &T::AccountId,
        domain: DomainId,
        responses: Vec<(QuestionHash, Answer)>,
    ) -> Result<bool, Error<T>> {
        let grid = TestGrids::<T>::get(domain)
            .ok_or(Error::<T>::GridNotFound)?;

        // Sample k random questions
        let k = T::QuestionsPerVerification::get();
        let seed = Self::random_seed();
        let selected = Self::sample_questions(&grid.questions, k, seed);

        // Verify responses match selected questions
        ensure!(
            responses.len() == selected.len(),
            Error::<T>::WrongQuestionCount
        );

        // Check answers
        let correct_count = responses.iter()
            .zip(selected.iter())
            .filter(|((q_hash, answer), expected_q)| {
                q_hash == *expected_q &&
                Self::verify_answer(*q_hash, answer)
            })
            .count();

        let score = (correct_count * 100 / k as usize) as u8;
        let passed = score >= grid.passing_threshold;

        // Record attempt
```

```
        EngagementAttempts::<T>::insert(
            (who, domain),
            EngagementRecord {
                score,
                passed,
                timestamp: Self::now(),
            }
        );

        Ok(passed)
    }

    pub fn slash_curator(
        curator: &T::AccountId,
        domain: DomainId,
        violation: ViolationType,
    ) -> DispatchResult {
        TestGrids::<T>::try_mutate(domain, |maybe_grid| {
            let grid = maybe_grid.as_mut()
                .ok_or(Error::<T>::GridNotFound)?;

            let stake = grid.curators.iter()
                .find(|(c, _)| c == curator)
                .map(|(_, s)| *s)
                .ok_or(Error::<T>::NotCurator)?;

            // Slash based on violation type
            let slash_amount = match violation {
                ViolationType::Inaccessibility => stake / 4,
                ViolationType::Ambiguity => stake / 2,
                ViolationType::Irrelevance => stake,
            };

            T::Currency::slash(curator, slash_amount);

            // Remove curator if fully slashed
            if slash_amount >= stake {
                grid.curators.retain(|(c, _)| c != curator);
            }

            Ok(())
        })
    }
}
```

D.4 Solidity Interface: EVM Compatibility

Listing 4: Solidity interface for cross-chain integration

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

interface IAugmentedDemocracy {
```

```
    struct Proposal {
        bytes32 id;
        address proposer;
        bytes32 contentHash;
        uint256 createdAt;
        uint256 votingEnds;
        ProposalStatus status;
    }

    enum ProposalStatus {
        Pending,
        Active,
        Passed,
        Rejected,
        Expired
    }

    struct Vote {
        address voter;
        bool approve;
        uint256 weight;
        uint8 entropy;
    }

    event ProposalCreated(
        bytes32 indexed id,
        address indexed proposer,
        bytes32 contentHash
    );

    event VoteCast(
        bytes32 indexed proposalId,
        address indexed voter,
        bool approve,
        uint256 weight
    );

    event ProposalResolved(
        bytes32 indexed id,
        ProposalStatus status,
        uint8 coherenceScore
    );

    // View functions
    function getCredential(address account)
        external view returns (uint256 reputation);

    function getCoherence(bytes32 proposalId)
        external view returns (uint8 gamma);

    function checkDualCondition(bytes32 proposalId)
        external view returns (bool passes);

    // State-changing functions
```

```
    function submitProposal(
        bytes32 contentHash,
        bytes32[] calldata artifacts
    ) external returns (bytes32 proposalId);

    function castVote(
        bytes32 proposalId,
        bool approve,
        bytes calldata engagementProof
    ) external;

    function resolveProposal(bytes32 proposalId) external;
}
```

D.5 Integration Patterns

D.5.1 Oracle Integration for Artifact Verification

Listing 5: Chainlink-style oracle for fact artifacts

```
pub trait ArtifactOracle {
    fn verify_artifact(
        artifact_hash: Hash,
        expected_issuer: AccountId,
    ) -> Result<ArtifactStatus, OracleError>;

    fn check_relevance(
        artifact_hash: Hash,
        proposal_domain: DomainId,
    ) -> Result<bool, OracleError>;
}

#[derive(Encode, Decode, Clone, TypeInfo)]
pub enum ArtifactStatus {
    Valid {
        issuer: AccountId,
        timestamp: BlockNumber,
        content_type: ContentType,
    },
    Invalid(InvalidReason),
    Pending,
}

impl<T: Config> Pallet<T> {
    pub fn validate_proposal_artifacts(
        proposal: &Proposal<T>,
    ) -> Result<(), Error<T>> {
        for artifact_hash in &proposal.artifacts {
            let status = T::Oracle::verify_artifact(
                *artifact_hash,
                proposal.expected_issuer.clone(),
            ).map_err(|_| Error::<T>::OracleError)?;

```

```
            match status {
                ArtifactStatus::Valid { .. } => {
                    // Check relevance
                    let relevant = T::Oracle::check_relevance(
                        *artifact_hash,
                        proposal.domain,
                    ).map_err(|_| Error::<T>::OracleError)?;

                    ensure!(relevant, Error::<T>::IrrelevantArtifact);
                }
                ArtifactStatus::Invalid(reason) => {
                    return Err(Error::<T>::InvalidArtifact(reason));
                }
                ArtifactStatus::Pending => {
                    return Err(Error::<T>::ArtifactPending);
                }
            }
        }

        Ok(())
    }
}
```

D.5.2 Quantum Entropy Integration

Listing 6: Quantum random number integration

```
pub trait EntropySource {
    fn get_entropy() -> [u8; 32];
    fn get_entropy_with_proof() -> (u8, EntropyProof);
}

impl<T: Config> Pallet<T> {
    pub fn sample_vote_entropy(
        voter: &T::AccountId,
        proposal_id: T::Hash,
    ) -> (u8, EntropyProof) {
        // Use quantum entropy if available, fall back to VRF
        if T::QuantumSource::is_available() {
            T::QuantumSource::get_entropy_with_proof()
        } else {
            // Fallback to verifiable random function
            let seed = Self::derive_seed(voter, proposal_id);
            let (random, proof) = T::VRF::generate(seed);

            // Extract single byte for epsilon
            let epsilon = random[0];
            (epsilon, EntropyProof::VRF(proof))
        }
    }

    pub fn weight_with_entropy(
        base_weight: u128,
```

```
        entropy_byte: u8,
    ) -> u128 {
        // Map [0, 255] to [-0.1, 0.1]
        // entropy_byte / 255 * 0.2 - 0.1
        let normalized = entropy_byte as i128 - 128; // [-128, 127]
        let epsilon = normalized * base_weight as i128 / 1280; // ~10%

        (base_weight as i128 + epsilon).max(0) as u128
    }
}
```

E Deployment Checklist

E.1 Pre-Deployment Requirements

1. **Identity Infrastructure**
   - DID registry deployed and operational
   - Identity verification oracles configured
   - Initial credential distribution mechanism ready
2. **Entropy Sources**
   - Primary: Quantum random beacon integration
   - Fallback: VRF with threshold signatures
   - Monitoring: Entropy quality metrics
3. **Test Grid Initialization**
   - Initial curator set selected
   - Domain taxonomy defined
   - Question pools seeded
   - Slashing parameters configured
4. **Parameter Configuration**
   - Epoch length (recommend: 28 days)
   - Coherence threshold (recommend: 50)
   - Engagement threshold (recommend: 70%)
   - Decay rate (recommend: 5%)

E.2 Security Audit Checklist

1. Formal verification of coherence calculation
2. Economic model review (token economics, attack costs)
3. Cryptographic review (signature schemes, hash functions)

4. Oracle trust assumptions documented
5. Upgrade mechanism security
6. Emergency pause functionality
7. Key management procedures
8. Incident response plan

E.3 Monitoring and Alerting

1. **Coherence Anomalies**
   - Alert: $\gamma < 30$ for any proposal
   - Alert: γ variance across proposals $> 2\sigma$
2. **Participation Metrics**
   - Track: Active credentials per epoch
   - Track: Vote distribution entropy
   - Alert: Sudden credential concentration
3. **Economic Health**
   - Track: Curator stake concentration
   - Track: Slashing frequency
   - Alert: $> 10\%$ curators slashed in epoch

F Reference Materials

F.1 Related Standards

- **W3C DID**: Decentralized Identifier specification
- **W3C VC**: Verifiable Credentials data model
- **EIP-712**: Typed structured data hashing
- **EIP-4824**: DAO URIs (governance metadata)
- **Substrate FRAME**: Pallet development framework

F.2 Mathematical Notation Reference

Symbol	Meaning
$\mathcal{P}$	Participant space
Π	Proposal space
$\mathcal{A}$	Artifact space
G	Test grid
$r(p,t)$	Credential at epoch t
$w(p,\pi)$	Vote weight
ϵ	Entropy injection
γ	Coherence score
σ_η^2	Entropy-weighted variance
λ	Decay rate
τ	Engagement threshold

Table 2: Notation reference